虚幻引擎知识地图（持续更新）

+ 收藏专题

钟钟(钟巧) 等 2022.08.10 11:57 15972 1084 142个资源

虚幻，一款非常优秀的商业引擎，市面上的很多3A大作均采用虚幻。在公司的战略转向3A大作的情况下，UE4是很多产品的首选引擎。本专题精选平台相关资源，给你一站式的了解~

推荐资源 站内分享 用手机查看 引用 投稿 分享至POPO眼界大开

在UE4上使用硬件Instance绘制动画模型

曾鹏程 2019.07.03 16:36 1701 28 2 查看原文

> 本文介绍在UE4上如何将骨骼动画信息烘焙到贴图，用渲染Instance静态模型代替渲染传统的骨骼模型，从而实现合批绘制3D动画角色，并包含Lod功能。

1 1.前言

G107(全面战争*战锤)的目标是在移动端打造3A级千人同屏3D手游。使用UE4自带的pbr材质和全局光照，在保证画面品质的前提下还要在移动端实现不锁视角的千人同屏，这在性能上还是有很大技术挑战的。

由于不锁视角，纸片人在中近距离和顶视角会穿帮，也无法跟环境光照融合，各种效果达不到策划要求。我们毅然抛弃了纸片人，选择了带LOD的3D模型。 在远距离时我们的LOD模型很简陋也不使用复杂的pbr，但是它有法线，可以在顶点计算简单Phong光照让LOD切换不那么突兀并能融入场景。但

2.1 制作资源转换插件。

此工具使用UE4插件的形势开发，主要功能是将美术制作的骨骼模型转化成静态模型，动画数据转化成贴图。
此工具的使用文档见UE4_GPU Instance静态SM资产生成

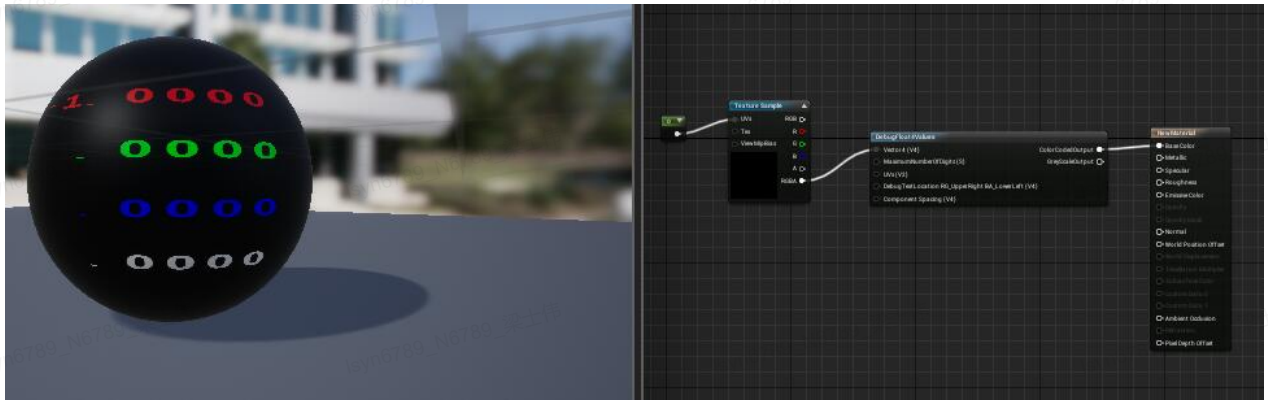
2.1.1 将骨骼模型转化成静态模型

通用的方法是将基础蒙皮后的顶点的位置，法线，UV等信息存在静态模型里。这里多做了一件事：就是将顶点受影响的骨骼信息存在第二套UV中，为了后续的蒙皮效率我们目前只存了一个权重最大的骨骼idx。因为如果要存两根骨骼idx，就需要两套UV, U表示骨骼idx, V表示骨骼权重。目前我们只用了U表示骨骼idx, V可以用来存储顶点色或mask信息，用于后续的变色等功能，让一个批次的模型有不同的样式。
如下图所示：左边是美术制作的骨骼模型（UV数是1），，右边是转换后静态模型（UV数是2）。

相关伪码

```
ConvertSkeletalMeshCptToRawMesh(USkeletalMeshComponent* skeMeshCpt...)
{
    for（逐lod处理）
    {
     skeMeshCpt->GetCPUSkinnedVertices(finalVertices, lodIndexRead);
     //拷贝finalVertices顶点位置信息到rawMesh
     for (int vertexIndex = 0; vertexIndex < finalVertices.Num(); ++vertexIndex)
        {                rawMesh.VertexPositions.Add(componentToWorld.TransformPosition(finalVertices[vertexIndex].Position));
        }
    处理submesh情况
    copy softVertices中权重最大的骨骼idx到rawMehs.WegetTexCoords[SECOND_UV_INDEX]
    从骨骼模型copy其他信息到rawMesh（如三角形信息）。
    }
```

此功能就是把每个动画的每一帧的所有骨骼转换矩阵存储在贴图里。为了达到美术效果，我们一个模型支持80根骨骼，所有动作总时长支持20秒，游戏是30帧每秒，那么我们最多需要存储80 * 20 * 30 = 48000根骨骼的矩阵转换信息，由于一个转换矩阵需要4*3个16位浮点表示（最后一个向量固定是0,0,0,1)。那么需要48000 * 4 * 3 * 16 / 8 = 1.1MB，也就是一个角色的所有动画数据加起来一般不超过1MB。这是我们完全能够接受的，所以我们没有使用两个四元数存储旋转和位置信息的方案，四元数方案若在shader中还原出矩阵会多出有一定开销。
如下图所示：我们采样动画贴图的一个像素（16bit * 4）debug输出到材质球向量值是（-1, 0, 0, 0)。



相关伪码

```
BakeAllAnimSequenceToTexture(TArray<UAnimSequence*>& arrayAnimSequence, ....)
{
    使用TSF_RGBA16F创建Texture
    for(每一帧)
    {
        for（每一个骨骼）
        {
            用自带的FFloat16编码矩阵中的浮点数
            把Float16存储到贴图。
            贴图的UV中U向代表骨骼index，V向代表时间，这样存储是为了方便shader取数据，会有一定的空白区域浪费内存。
        }
    }
}
```

2.2 核心实现：修改UE4源码。

按Instancing的方式绘制静态模型几乎是所有引擎都支持的功能，植被几乎都采用这种方式，这也是最最节省的一种合批绘制策略，因为并不涉及到顶点和贴图合并，所有的Instance都是共用一份顶点数据和贴图，硬件从底层就支持此绘制方式，OpenGL的相关渲染命令有：
`glDrawArrayInstanced, glDrawElementsInstanced, glVertexAttribDivisor等 //详情可以参考《OpenGL编程指南》 第八版3.5多实例渲染`

里由于数量过多，使用了VertexBuffer).打包每个Instance的Custom数据，在材质里通过InstanceIdx索引出自己需要的Custom数据。它跟顶点数据和贴图（不需要经常更新）不一样，它的更新频率较高,所以需要使用Dynamic的VertexBuffer。

```
void FStaticMeshInstanceBuffer::InitRHI()
{
    ...
    CreateVertexBuffer(InstanceData->GetCustomResourceArray(), BUF_Dynamic , ...)
    ...
}
```

UE4Instance相关buffer是不支持更新的，需要增加更新VertexBuffer功能,

```
void FStaticMeshInstanceBuffer::UpdateVertexBuffers()
{
    ...
    SizeInBytes = InstanceData->GetCustomResourceArray()->GetResourceDataSize();
        VertexBufferData = RHILockVertexBuffer(InstanceCustomBuffer.VertexBufferRHI, 0, SizeInBytes, RLM_WriteOnly);
        FMemory::Memcpy(VertexBufferData, InstanceData->GetCustomResourceArray()->GetResourceData(), SizeInBytes);
        RHIUnlockVertexBuffer(InstanceCustomBuffer.VertexBufferRHI);
    ...
}
```

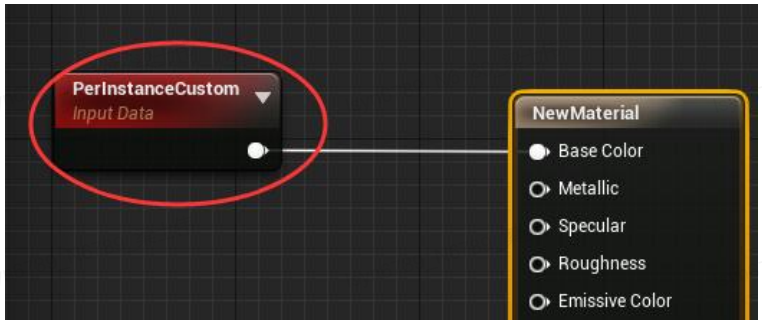
UE4的instance基本是Static的，用于植被渲染，所以没有考虑过更新问题，所有更新instance的操作都是销毁渲染数据然后重新创建（在引擎中就是重新创建新的SceneProxy）。我们不可能更新动作或者位置就销毁数据重新传递新的，这样显然会有严重的性能问题并且会引起闪烁。我们需要增加在逻辑层更新RHI（渲染硬件接口）InstanceVertexBuffer的功能。此功能的代码比较多，就不一一贴出来了，下面的代码只是在逻辑县城如何更新RHI数据调用的地方。代码中可以发现我们使用了UHierarchicalInstancedStaticMeshComponent作为基类，这是因为我们希望Instance模型有Lod功能，也就是在CPU计算出Lod分层信息，远处的模型依然可以使用低面Lod, 同一个兵种如果离镜头远近幅度较大，Lod相同的模型会合批绘制。这样可以减少GPU的计算压力。

```
if (IsAniInstance() && SceneProxy)
                {
                        if (!IsRenderStateDirty())
                        {
                                MarkRenderTransformDirty();

                                int32 FirstUnbuiltIndex = NumBuiltInstances > 0 ? NumBuiltInstances : NumBuiltRenderInstances;
                                auto ClusterTreePtrRef = ClusterTreePtr.ToSharedRef();
                                FMatrix CompTransfrom = GetComponentTransform().ToMatrixWithScale();
                                int32 InInstanceCountToRender = InstanceCountToRender;
```

```
                                        {
                                                ThisSceneProxy->Reset_RenderThread(ClusterTreePtrRef, FirstUnbuiltIndex,
InstanceCountToRender, InOcclusionLayerNumNodes, CompTransfrom); //这里是更新原来的SceneProxy而不是重新创建。
                                        });
                                }
                        }
```

另外除了为逻辑线程提供更新VertexBufferr的接口外，还要为材质编辑提供支持，这样所有想用这个功能的材质都可以受益。如下图所示是材质编辑器新增的一个节点，PerInstanceCustom节点：



在UE4增加自定义材质节点网上有比较多教程，大家可以搜索参考，需要注意的是，由于我们使用VertexBuffer传递，出于性能考虑，perinstance的只在vertexshader里访问才有效，并没有传递到pixshader.由于我们主要用作顶点变换可以满足需求，如果需要在pixshader中使用可以通过VS->PS的out-in方式自行传递。

下面贴一下材质模板两个比较重要文件的修改，一个数据的定义和接收，一个是材质节点对应的源码函数。

在LocalVertexFactory.ush文件中数据的定义和接收：

```
struct FVertexFactoryIntermediates

{

...

# if USE_INSTANCING

    float4 PerInstanceCustom;

# endif
```

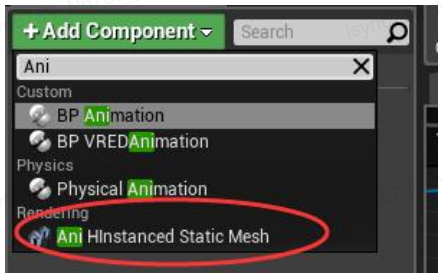
```
# if MANUAL_VERTEX_FETCH
        ...
        Buffer InstanceCustomBuffer;
        ...
# endif
};
...
FMaterialVertexParameters GetMaterialVertexParameters(FVertexFactoryInput Input,...)
{
...
# if USE_INSTANCING && MANUAL_VERTEX_FETCH
        Intermediates.PerInstanceCustom = InstanceCustomBuffer[(InstanceId + InstanceOffset)];
# endif
...
}
```

MaterialTemplate.ush中定义的材质节点对应的源码函数：

```
float4 GetPerInstanceCustom(FMaterialVertexParameters Parameters)
{
        # if USE_INSTANCING
                return Parameters.PerInstanceCustom;
        # else
```

2.2.2 支持逐Instance用户数据的组件。

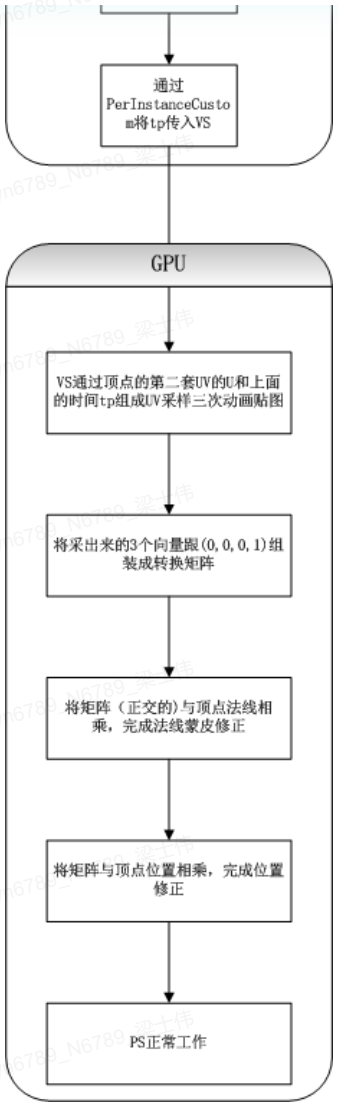
我们项目主要是实现动画Instance功能，并且还需要按LOD分级合批绘制，所有我们新增了AniHInstancedStaticMesh组件。在UE编辑器可直接添加这个组件（如下图所示）：



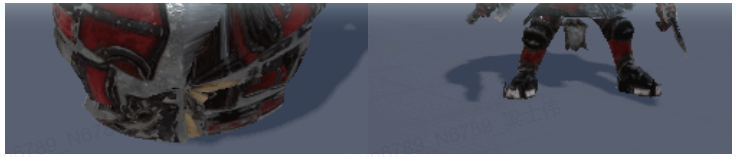
简单编辑一下就可以看到PerInstanceCustom的效果， 如下图所示的两个Actor，左边的方阵Actor是添加功能之前的，如草木般，而右边的Actor使用AniHInstancedStaticMesh组件，每个士兵都有不同的姿态，他们是一个批次绘制完成的。

2.3 制作通用动画Instance材质。

我们的功能主要是在顶点Shader完成，原理跟GPU蒙皮差不多。
具体实现的流程图如下：



很明显opengles 2.0对这套方案不支持，因为它需要在顶点shader进行纹理采样，但是3A作品不支持es2.0也无可厚非。 通过此材质，我们的静态模型就会按动画的方式动起来。如下图所示，前面是材质球动图，后面是应用前面材质的静态模型：



3 3.总结

3.1 统计数据

虽然上面的方案只支持同样的模型之间的合批，但收益依然是巨大的，原来绘制100人的军团至少需要100个 批次 ，而10个这样的军团就要1000个批次，这在移动端是无法接受的。而我们项目恰好符合这种重复使用造型的特性。不考虑LOD时，我们项目十个军团的小兵10个批次就可以绘制完成。如下图的统计数据所示：1156个instance在一个军团里，总面数是402W，在移动端渲染也是没有问题的。

3.2 应用

此方法不仅为小兵提供支持，还可以丰富UE4的植被表现，因为每一棵树都有自己的custom数据，为美术提供了发挥空间。

另外一个应用就是代替某些粒子特效， 比如在我们项目中集体放箭不再使用粒子特效模拟，而是instance, 这样弓箭兵可以点对点的射出箭矢，让画面更加真实。

下面的动图是综合应用展示，所有小兵都是Instance合批绘制, 所有的箭矢也是用Instance合批绘制，每个箭矢在不同抛物线阶段都有不同曲度的拖尾。

3.3 参考资料：

https://blogs.unity3d.com/cn/2018/04/16/animation-instancing-instancing-for-skinnedmeshrenderer/

全部评论 2

请输入评论内容

最热 最新

HansenT(谭国瀚)

2楼 曾帅 人帅业务强

2022-02-17 16:43 回复 0

匿名

1楼 非常棒，谢谢分享~

2019-11-04 20:03 回复 0

加载完毕,没有更多了